Quick Performance Bounds for Computer and Storage Systems with Parallel Resources

Elizabeth Varki Larry Dowdy Chang Zhang

Abstract

This paper presents a quick single-step performance bounding technique for parallel computer and storage systems. The computation of these bounds is so simple that it can even be performed by hand. The bounding technique develops balanced job bounds and asymptotic bounds for parallel systems. These bounds are the parallel system counterparts to previously developed bounds for serial product-form systems. The contribution of this paper is showing how these simple bounds for serial networks can be applied to parallel networks. The performance technique is potentially useful in the early phases of design projects or capacity planning studies when several candidate configurations have to be evaluated quickly.

Index Terms: performance evaluation, parallel computer and storage systems, fork-join networks, queueing systems, performance bounds.

1 Introduction

The performance of computing and storage systems can be improved by parallelism. In computing systems, the execution time of a program can be improved by splitting the program into sub-programs which then execute in parallel on different processors. In storage systems, the service time of an I/O request can be improved by dividing the request into sub-requests which then execute in parallel on multiple disks. From a performance analysis perspective, such parallel resources are modeled by *fork-join* queues, since a job *forks* into its various sub-tasks and then *joins* (exits) after all these sub-tasks complete service. For ease of exposition, we will henceforth use the term parallel networks to refer to systems with parallel computing or parallel storage resources.

This paper presents a simple, yet powerful technique for computing the mean performance bounds of parallel networks with batch and terminal workloads (see Figure 1). The technique exploits results on balanced job bounds for product-form, serial networks in [34]. The principle that the performance of a product-form network is bounded by the performance of suitably balanced networks also applies to parallel networks. In [34], simple expressions for computing the performance of balanced product-form networks are provided, but unfortunately these results do not carry over to parallel networks. In this paper, however, we derive simple expressions for computing the approximate performance measures of *balanced parallel* networks by using an approximate response time result in [32] and Theorem 4.1 that is proven in this paper. (Comparison of these approximations against simulations result in an average error of 3% for balanced parallel networks with multiprogramming levels ranging from 1 to 50 and degree of parallelism ranging from 1 to 50.) This expression is then used to compute approximate optimistic and pessimistic bounds for any parallel network. Since the results in [32] are proven only for exponential service time distributions, the balanced job bounds are valid only for exponential service time distributions. In addition, we derive asymptotic performance bounds for parallel networks using the idea in [24] that the performance of a serial network is bounded above and below by the performance of the network under light and heavy workloads, and these bounds, which are easily computed, are independent of the service time distribution of the servers. Thus, performance engineers can compute both types of bounds (balanced and asymptotic) and choose the tighter of these two bounds. The value of these performance bounds is that the approximate performance bounds of parallel networks can be derived using a single-step computation,

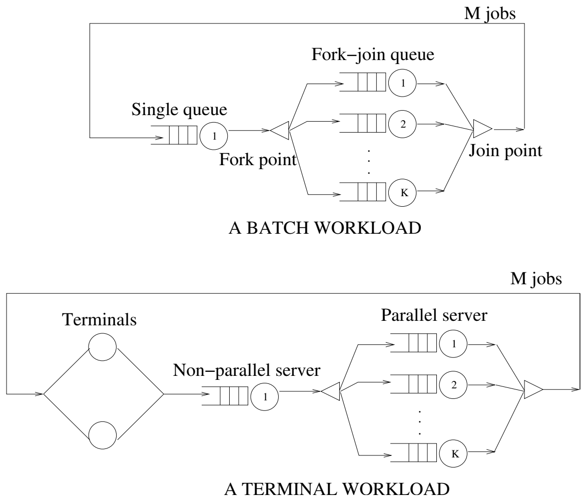

Figure 1: Examples of parallel networks

regardless of the degree of parallelism, the load on the system, and the number of modeled resources. Thus, this bounding technique is simple and efficient, with the added advantage of requiring a single-step to calculate (as opposed to the iterative calculations required by alternative approaches).

Our technique can be used in the early stages of a design or capacity planning project when the inputs are not completely known and/or when a large number of candidate configurations must be evaluated - as in automated storage management systems such as Minerva [2] that must evaluate thousands of candidate storage configurations quickly. Subsequent to the elimination of non-suitable configurations, it is recommended that performance analysts use more precise, but computationally difficult bounds/approximations (*e.g.*, detailed simulations).

The remainder of this paper is organized as follows. Section 2 summarizes related work on performance analysis of fork-join systems. Section 3 presents a prior result on parallel networks from [32] that is used here to derive the quick bounds. Sections 4 and 5 develop the balanced job bounds and the asymptotic bounds, respectively. Section 6 illustrates the application of the bounds to disk arrays. Finally, conclusions are presented in Section 7.

2 Related Work

Several papers study parallel (fork-join) queues and propose tools for analyzing their performance. Exact performance measures have been derived only for fork-join queues with two servers - [3] and [11] derive exact steady state distribution for two server fork-join queues in open networks while [32] derives exact mean performance measures of two server fork-join queues in closed networks. Results in [3] assumes general service time distribution, while those in [11] and [32] assume exponential service time distributions. Due to the difficulty of analyzing fork-join queues exactly, all studies on fork-join queues with three or more servers concentrate on approximation or bounding techniques. Heidelberger and Trivedi [12] consider a closed queuing network in which jobs divide into two or more asynchronous tasks. The join point is not modeled. The service centers are of a type described in the BCMP theorem [9]. They develop an iterative method for solving a sequence of product-form models. In [13], the model is expanded to include a join node. Nelson and Tantawi [25] consider a scaling approximation technique to analyze the mean response time of an open homogeneous fork-join queue with exponential service time distributions. They assume that the mean response time increases at the same rate as the number of sibling tasks. Closed-form approximation expressions for the mean response time are developed. An extension of this approximation to heavy traffic, relying on a light traffic interpolation technique, is developed by Makowski and Varma [23]. Kim and Agrawala [15] analyze waiting times for two server open, homogeneous fork-join systems with exponential and 2-stage Erlang service time distributions. In [21, 22], Lui, Muntz, and Towsley present a bounding technique for an open, homogeneous fork-join network with a k-stage Erlang distribution. Liu and Perros [19, 20] propose an approximation procedure based on decomposition and aggregation for analyzing a closed queuing system with K-sibling fork-join queues. Their method provides an upper bound for mean response time. Response time bounds are obtained for acyclic fork-join queuing networks by Baccelli et. al. [4] using stochastic ordering principles and association of random variables. Baccelli and Liu [5] propose a new class of queuing models for evaluating the performance of parallel systems. Using the concept of associated random variables, Kumar and Shorey [16] obtain response time bounds for an open fork-join model in which a job forks into a random number of tasks. Service times are drawn from a general distribution. Almeida and Dowdy [1] propose an iterative technique for obtaining lower performance bounds of closed fork-join networks with exponential service times. No proofs for the technique are presented. Varki and Dowdy [30] prove that the proportion of the number of jobs in the different subsystems of a closed, exponential, balanced fork-join network remain constant irrespective of the multiprogramming level. This property of balanced fork-join networks is used to bound the performance of fork-join networks. The proof is limited to 2-server exponential fork-join systems. In [31], Varki develops an approximate mean-value analysis technique for fork-join parallel networks. Balsamo, Donatiello, and Van Dijk [8] propose a matrix-geometric algorithmic approach for computing performance bounds of open heterogeneous fork-join systems. The fork-join structure is studied with relation to parallel storage systems (RAID) in [18, 26, 27, 28].

All the above performance techniques grow in complexity as the degree of parallelism increases (i.e., as the number of servers in the fork-join queue increases) and as the number of jobs accessing the fork-join queue increases. Most of these techniques do not address the tightness of their generated bounds/approximations. The contribution of this work is the computational speed and the simplicity of the bounding technique presented. Regardless of the complexity of the parallel network or the load on the network, the bounds can be computed by simple non-iterative formulae. These bounds accurately quantify the effects of bottleneck devices, a critical factor in capacity planning studies. So, these bounds are potentially useful in the early phases of a design or capacity planning project.

3 Prior Result on Parallel Networks

We present a result on mean response time of parallel fork-join queues from [32] that is used here to develop quick performance bounds. Before presenting the result, we present the model and notation. Consider parallel networks with batch or terminal workloads, as shown in Figure 1. Let K represent the number of queuing resources, both parallel and non-parallel, in a network. It is assumed that all service centers of a parallel resource are identical and that all jobs in the networks are identical (i.e., the workload is single-class). Jobs in the network will access each of the K resources with some probability. For each resource k, V_k represents the probability that a job will visit (access) resource k during each cycle and v_k represents the probability that a sub-task of a job visits a service center of the resource during each cycle. For non-parallel resources, $v_k = V_k$, and for parallel resources, $v_k = V_K * p$ where $p > 0$. If S_k is the mean service time at resource k, the service demand at resource k, D_k, equals $V_k * S_k$. Similarly, the service demand, d_k, at a service center within a parallel resource is given by $v_k * s_k$, where s_k is the mean service time at the center. Performance techniques typically compute the mean performance measures (throughput, response time, queue length) of each resource (X_k, R_k, Q_k) and of the overall network (X, R, Q). Let M represent the number of jobs cycling through the network. To refer to a performance parameter at a specific multiprogramming level M, the symbol (M) is attached to the parameter.

First, consider fork-join queues with N service centers where every arriving job forks into exactly N sub-tasks. Thus, when a job "visits" the fork-join queue, a sub-task of the job will "visit" each service center of this fork-join queue (i.e., $V_k = v_k, p = 1$). In addition, assume that the service times at the N service centers of the fork-join queue are exponentially distributed. Then, the response time, R_k, of the fork-join queue is bounded by

$$R_k \leq D_k + d_k * A_k$$

Here, $D_k = V_k * S_k$ is the mean service demand at the fork-join queue. The mean service time S_k of a fork-join queue with exponential service times is given by

$$S_k = H_N * s_k$$

where H_N is the N^{th} harmonic number (*i.e.*, $H_N = \sum_{i=1}^{N} 1/i$) and s_k is the mean service time at a center within the fork-join queue. The parameter V_k is the probability that a job in the network visits resource k during each cycle, while v_k is the probability that a sub-task of a job visits a device of resource k. The parameter $d_k = v_k * s_k$ represents the mean service demand at a center within the fork-join queue. Finally, A_k is the mean number of jobs seen at the fork-join queue when a new job arrives at the queue. The relationship is a strict equality only in the case of closed fork-join networks with a single resource, namely, a fork-join queue with two service centers. In other cases, the computed response times are greater than the actual response times. Simulation results show that the computed values are close to the simulated actual values with the average error being 3% for values of N and M ranging from 1 to 50.

Next consider fork-join queues where jobs can split into any number of sub-tasks (i.e., jobs are not restricted to forking into exactly N sub-tasks; the number of sub-tasks can be either ¡ N or = N or ¿ N). So, V_k, the probability that a job visits a fork-join queue, is not necessarily equal to v_k, the probability that a sub-task of a job visits a center within the fork-join queue. Furthermore, there is no restriction on the service time distribution. In [32], an argument is presented showing that

$$\boxed{R_k \approx D_k + d_k * A_k} \tag{1}$$

Equation 1 approximates the response time at a resource to the sum of the service time (D_k) and the wait time ($d_k * A_k$) at the resource. Note that for non-parallel resources, $D_k = d_k$ and the above equation reduces to $R_k = d_k * (1 + A_k)$.

4 Balanced Job Bounds for Parallel Networks

Equation 1 is used here to compute single-step approximate performance measures of balanced parallel networks. A balanced parallel network is one where the demands, d_k, at all the service centers in the network are equal (i.e., $d_1 = d_2 = d_3 = \cdots = d_K$). The key idea underlying the balanced job bounding technique is that the performance of any given parallel network is bounded by the performance of two balanced parallel networks: (1) pessimistic bounds are obtained when the demands at all service centers are raised to the maximum demand at any service center in the network, and (2) optimistic bounds are obtained when the demands at all service centers are reduced to the minimum demand at any service center in the network. We show here that the approximate mean performance measures of a balanced network can be computed in a single step regardless of the number of resources and jobs in the network. Thus, single-step approximate performance bounds of parallel networks can be generated quickly and trivially.

The balanced job bounding technique developed here is the parallel network counterpart of the balanced job bounding technique for product-form networks [34]. In [34], the balanced job bounding technique is derived by using the well known fact that the mean queue lengths at all the resources of a balanced product-form network are equal at all multiprogramming levels. Thus, the mean throughput and response times of balanced product-form networks can be computed trivially using Little's Law. For balanced parallel networks, this property of equality of mean queue lengths does not hold since the mean queue length at resource k is dependent not only on the service demand d_k at the resource but also on the degree of parallelism at the resource. In the next section, we show how the mean performance measures of balanced parallel networks containing both parallel and non-parallel resources can be computed quickly. The work here is an extension of [30] where the bounding technique and analysis are limited to parallel networks with fork-join queues containing only two exponential service centers. In this work, the balanced job bounds are derived for parallel networks modeling parallel resources with $N > 1$ devices where arriving jobs can divide into any number of sub-tasks.

4.1 Balanced Job Bounds for Batch Workloads

We first explain how Equation 1 is used to derive quick single-step approximate performance measures of balanced parallel networks under batch workloads. For balanced networks, $d_1 = d_2 = \cdots = d_K$. Let d $(= d_1 = d_2 = \cdots = d_K)$ represent the service demand at a service center in a balanced parallel network. Then using Equation 1, the approximate response time, R, of a balanced parallel network is given by:

$$\begin{aligned} R &\approx (D_1 + D_2 + \cdots + D_K) + d * (A_1 + A_2 + \cdots + A_K) \\ &= D + d * (A_1 + A_2 + \cdots + A_K) \end{aligned}$$

where $D = D_1 + D_2 + \cdots + D_K$. The values of D (the sum of service demands at each resource) and d (the service demand at a service center) can be derived if the service time distribution and the configuration of the network are known. The issue now is to compute a value for $(A_1 + A_2 + \cdots + A_K)$ in a single step. Here, A_k represents the mean number of jobs at resource k seen by a job just prior to arrival at resource k. In the best case scenario, a job arriving at resource k sees no jobs ahead of it. In the worst case scenario, a job arriving at resource k seen $(M - 1)$ jobs ahead of it. Thus, $\mathbf{0} \leq (A_1 + A_2 + \cdots + A_K) \leq \mathbf{K} * (\mathbf{M} - \mathbf{1})$ is a loose bound that is intuitive and simple to derive. In the following theorem, we show through a non-trivial proof that the upper bound can be lowered from $K * (M - 1)$ to $(M - 1)$ and the lower bound can be increased from 0 to maximum$\{0, (M - K)\}$. The line of reasoning in the proof presented below is similar to that used by Burke [10] in proving the equality of arrival and departure instant probabilites.

Theorem 4.1 *For all closed queuing networks (not just parallel queuing networks) with K resources and M circulating jobs,*

$$\textit{maximum}\{0, M - K\} \leq A_1(M) + A_2(M) + \cdots + A_K(M) \leq M - 1$$

where A_k is the mean number of jobs at resource k that is seen by a job just prior to arrival at resource k.

Proof: Without loss of generality, assume that $V_1 = V_2 = \cdots = V_K$ so that jobs cycle through the network moving from resource 1 to resource K.

Let A represent the mean arrival instant queue length of the network. That is, A is the sum of the mean number of jobs at the K resources seen by a job just prior to its arrival at the network (*i.e.,* arrival at resource 1). For closed networks with M circulating jobs, $A(M) = M - 1$, since a job sees $M - 1$ jobs in the network just prior to arrival (*i.e.* excluding itself). In particular, let A^n represent the mean arrival instant queue length of the network just prior to the n^{th} cycle through the network; that is, if one counts the number of arrivals at a resource, A^n represents the queue length of the network just prior to the n^{th} arrival at resource 1. Now,

$$A^n(M) = A(M) = M - 1$$

Let J_k represent the number of jobs at resource k at a particular instant. Assume that just prior to the n^{th} arrival at the network (*i.e.,* arrival at resource 1), the state of the network is given by

$$J_1 = n_1, \quad J_2 = n_2, \quad J_3 = n_3, \quad .... \quad , J_K = n_K$$

where $n_1 + n_2 + n_3 + \cdots + n_K = M - 1$. At this instant just prior to the n^{th} arrival at resource 1, there have been:

$(n-1)$ arrivals at resource 1,

$(n-1-n_1)$ arrivals at resource 2,

$(n-1-n_1-n_2)$ arrivals at resource 3,

......

$(n-1-n_1-n_2-...-n_{K-1})$ arrivals at resource K.

We first show that $A_1(M) + A_2(M) + \cdots + A_K(M) \leq M - 1$ by looking at the arrival state at each of the resources just prior to the **next** arrival of a job at the resource.

Just prior to the n^{th} arrival at resource 1, the number of jobs at resource 1 will be equal to n_1 (*i.e.*, $J_1 = n_1$).

Just prior to the $(n-n_1)^{th}$ arrival at resource 2, the number of jobs at resource 2 will be less than or equal to n_2 (*i.e.*, $J_2 \leq n_2$), since some of the n_2 jobs may have completed service and departed resource 2 by the time that the $(n-n_1)^{th}$ job arrives at resource 2.

Similarly, for each resource k, just prior to the $(n-n_1-n_2-...-n_{k-1})$ arrival at resource k, $J_k \leq n_k$.

Thus,

$$A_1^n + A_2^{n-n_1} + A_3^{n-n_1-n_2} + \cdots + A_K^{n-n_1-...-n_{K-1}} \leq A^n$$

In the steady state, as n approaches infinity

$$\begin{aligned} lim_{n\to\infty}(A_1^n + A_2^{n-n_1} + A_3^{n-n_1-n_2} + \cdots + A_K^{n-n_1-...-n_{K-1}}) &= lim_{n\to\infty}(A_1^n + A_2^n + A_3^n + \cdots + A_K^n) \\ &\leq lim_{n\to\infty}A^n \end{aligned}$$

Now,

$$lim_{n\to\infty}A^n(M) = A(M) = M - 1$$

Thus,

$$A_1(M) + A_2(M) + \cdots + A_K(M) \leq M - 1$$

Using a similar approach, we now prove that $A_1(M) + A_2(M) + \cdots + A_K(M) \geq M - K$ by looking at the arrival state at each of the resources just prior to the **last** arrival of a job at the resource

Just prior to the n^{th} arrival at resource 1, the number of jobs at resource 1 is equal to n_1 (*i.e.*, $J_1 = n_1$).

Just prior to the $(n-1-n_1)^{th}$ arrival at resource 2, the number of jobs at resource 2 was greater than or equal to $n_2 - 1$ (*i.e.,* $J_2 \geq n_2 - 1$), since some of the jobs at resource 2 may have completed service and departed resource 2 by the time that n^{th} job arrives at the network (*i.e.,* arrives at resource 1).
Similarly, for each resource k, just prior to the $(n-1-n_1-n_2-...-n_{k-1})$ arrival at resource k, $J_k \geq n_k - 1$.
Thus,

$$A_1^n + A_2^{n-1-n_1} + A_3^{n-1-n_1-n_2} + \cdots + A_K^{n-1-n_1-...-n_{K-1}} \geq A^n - (K-1)$$

In the steady state, as n approaches infinity

$$\begin{aligned} lim_{n\to\infty}(A_1^n + A_2^{n-1-n_1} + A_3^{n-1-n_1-n_2} + \cdots + A_K^{n-1-n_1-...-n_{K-1}}) &= lim_{n\to\infty}(A_1^n + A_2^n + A_3^n + \cdots + A_K^n) \\ &\geq lim_{n\to\infty}A^n - (K-1) \\ &= (M-1) - (K-1) \\ &= M - K \end{aligned}$$

Thus,

$$M - K \leq A_1(M) + A_2(M) + \cdots + A_K(M) \leq M - 1$$

□

Theorem 4.1 states that the sum of arrival instant queue lengths at all resources of a closed network with K resources and M circulating jobs is at most equal to $M-1$ and at least equal to $M-K$. (In particular, for product form networks, $A_1 + A_2 + \cdots + A_K = M - 1$.) The theorem makes no assumptions regarding the service time distributions and the theorem holds for all networks, not just balanced parallel networks.

Using Theorem 4.1, the Equation $R \approx D + d * (A_1(M) + A_2(M) + \cdots + A_K(M))$ can be written as

$$\boxed{R(M) \approx D + d * (M-1)} \tag{2}$$

In the case of balanced parallel exponential networks, the response time is given by $R \leq D + d * (M-1)$ since Equation 1 is an inequality (See Section 3). We test the tightness of Equation 2 via simulation. Figure 2 shows three graphs that plot the response time of balanced parallel networks containing two resources, one parallel and the other non-parallel. The first graph relates to exponential servers, the second graph relates to Erlang servers with coefficient of variation (CV) [1] = 0.5, and the third graph relates to hyper-exponential servers with coefficient of variation = 1.5. Comparison of the response time approximations against simulations show an average error of 3% at 95% confidence level for exponential, Erlang, and hyper-exponential distributions with multiprogramming levels ranging from 1 to 50 and degree of parallelism ranging from 1 to 50.

The mean approximate throughput, X, of a closed balanced parallel network is derived from the response time equation using Little's Law.

$$\boxed{X(M) \approx \frac{M}{D + d * (M-1)}} \tag{3}$$

[1] CV = standard deviation/mean

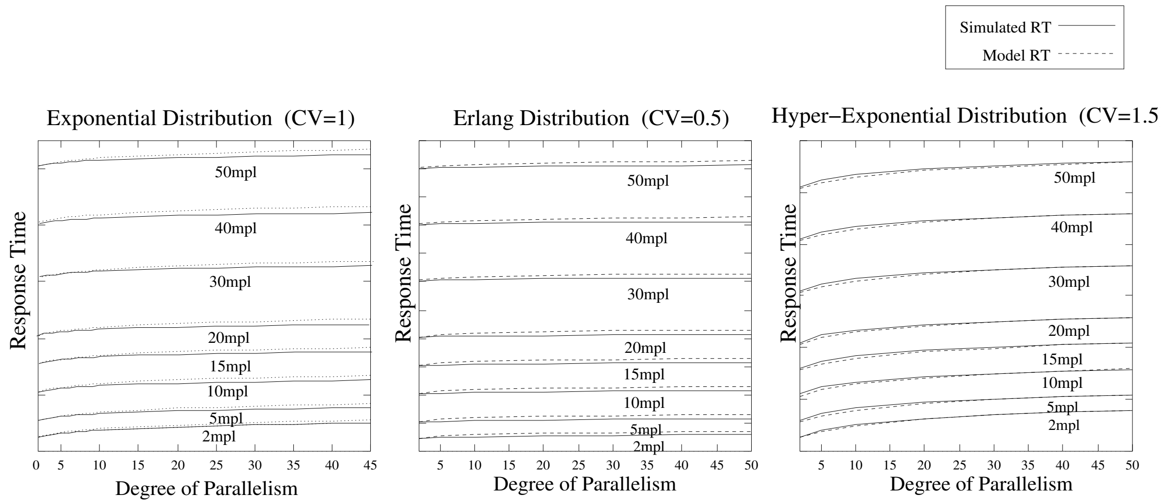

Figure 2: Response time of balanced parallel networks

The approximate performance measures of balanced parallel networks can be computed in a single step using Equations 2 and 3. Since these equations are derived from Equation 1, they are valid if the assumptions for the validity of Equation 1 hold, namely, that all fork-join queues in the parallel network have exponential service times.

The performance measures of any parallel network are bounded by the performance of two related balanced parallel networks, one in which the demands at all service centers are raised to d_{max}, the maximum demand at any service center of the parallel network, and another in which the demands at all the service centers are reduced to d_{min}, the minimum demand at any service center of the network. Thus,

$$\frac{M}{D_{max} + d_{max} * (M-1)} \approx\leq X(M) \approx\leq \frac{M}{D_{min} + d_{min} * (M-1)}$$

where $D_{max} = D_1 + D_2 + \cdots + D_K$ is computed by setting the service demand at all service centers to d_{max} and $D_{min} = D_1 + D_2 + \cdots + D_K$ is computed by setting the service demand at all service centers to d_{min}.

Tighter optimistic bounds can be obtained for networks under heavy load as explained here. At high multiprogramming levels, the first resource to saturate is the resource containing the bottleneck center with demand d_{max}. The utilization, u_{max}, of this center approaches the maximum limit of 1. Using the Utilization Law ($U = X * D$), an optimistic throughput bound for the parallel network under heavy load is given by

$$X(M) \leq \frac{u_{max}}{d_{max}} \leq \frac{1}{d_{max}}$$

At lower multiprogramming levels, the optimistic balanced bound computed using d_{min} is tighter. The optimistic balanced bound intersects the heavy load bound as the multiprogramming level increases to some point, say, m^*. After this point, the heavy load bound is tighter. This gives

$$\boxed{\frac{M}{D_{max} + d_{max} * (M-1)} \approx\leq X(M) \approx\leq \text{minimum}\left\{\frac{M}{D_{min} + d_{min} * (M-1)}, \frac{1}{d_{max}}\right\}}$$

The response time bounds are computed from the throughput bounds by using Little's Law.

$$\boxed{\text{maximum}\{D_{min} + d_{min} * (M-1),\ M * d_{max}\} \approx\leq R(M) \approx\leq D_{max} + d_{max} * (M-1)}$$

4.2 Balanced Job Bounds for Terminal Workloads

For terminal workloads, a job in the network spends time Z at a terminal (modeled by a delay server) during each cycle. The cycle time of the balanced network is given by

$$Z + R(M) \approx Z + (D + d * (A_1 + A_2 + \cdots + A_K))$$

The issue now is to compute an approximate value for $(A_1 + A_2 + \cdots + A_K)$ in a single step. For batch workloads, the number of waiting jobs $(A_1 + A_2 + \cdots + A_K)$ is approximately equal to $(M - 1)$ since jobs spend all their time at queuing resources. For terminal workloads, the number of waiting jobs at the queuing resources $(A_1 + A_2 + \cdots + A_K)$ would be less than $(M - 1)$ since some jobs may be at the terminals. In order to generate quick bounds, a tighter bound/approximation for $(A_1 + A_2 + \cdots + A_K)$ is required. The approach presented here is similar to the approach used to compute an approximate value of $(A_1 + A_2 + \cdots + A_K)$ for product-form networks under terminal workloads [17].

The degree by which $(A_1 + A_2 + \cdots + A_K)$ varies from $(M - 1)$ depends on the relative time that each job spends at the queuing resources as opposed to the time it spends at the terminal server. This gives:

$$A_1(M) + A_2(M) + \cdots + A_K(M) \approx \frac{R(M)}{R(M) + Z} \times (M - 1)$$

Bounds for $R(M)$ are computed as follows:

- In the worst-case scenario, each arriving job has to wait behind $(M - 1)$ jobs at each of the queueing resources before receiving service. Let $D = D_1 + D_2 + \cdots + D_K$. The job spends $D * (M - 1)$ time units waiting for service, D time units receiving service, and Z time units thinking. Therefore, the relative time spent by a job in the queueing resources is given by $\frac{M*D}{M*D+Z}$ and $A_1(M) + A_2(M) + \cdots + A_K(M) \approx \frac{M-1}{1+Z/(M*D)}$.
- In the best-case scenario, each arriving job receives service immediately (i.e., there are no waiting jobs). The job spends D time units in service and Z time units thinking. Therefore, the relative time spent by a job at the queuing resources is given by $\frac{D}{D+Z}$ and $A_1(M) + A_2(M) + \cdots + A_K(M) \approx \frac{M-1}{1+Z/D}$.

The throughput of a parallel network under a terminal workload is given by $X(M) = M/(Z + R(M))$. The balanced job bounds for a parallel network under a terminal workload are then computed as:

$$\boxed{\frac{M}{Z + D_{max} + \frac{d_{max}*(M-1)}{1+Z/(M*D_{max})}} \approx\leq X(M) \approx\leq \text{minimum}\left\{\frac{M}{Z + D_{min} + \frac{d_{min}*(M-1)}{1+Z/D_{min}}}, \frac{1}{d_{max}}\right\}}$$

The approximate response time bounds are computed from the throughput bounds by using Little's Law.

$$\boxed{\text{maximum}\left\{D_{min} + \frac{d_{min} * (M-1)}{1 + Z/D_{min}},\ M * d_{max} - Z\right\} \approx\leq R(M) \approx\leq D_{max} + \frac{d_{max} * (M-1)}{1 + Z/(M * D_{max})}}$$

4.3 Validation

Figure 3 plots the balanced job bounds for networks with exponential servers. The performance measures of parallel networks are generated via simulation. The simulated network for batch workloads has 4 resources - resource 1 is non-parallel with mean service time of 1 time unit, resource 2 is a parallel queue with 2 service centers and mean service time of 2 time units, resource 3 is a parallel queue with 3 service centers and mean service time of 3 time units,

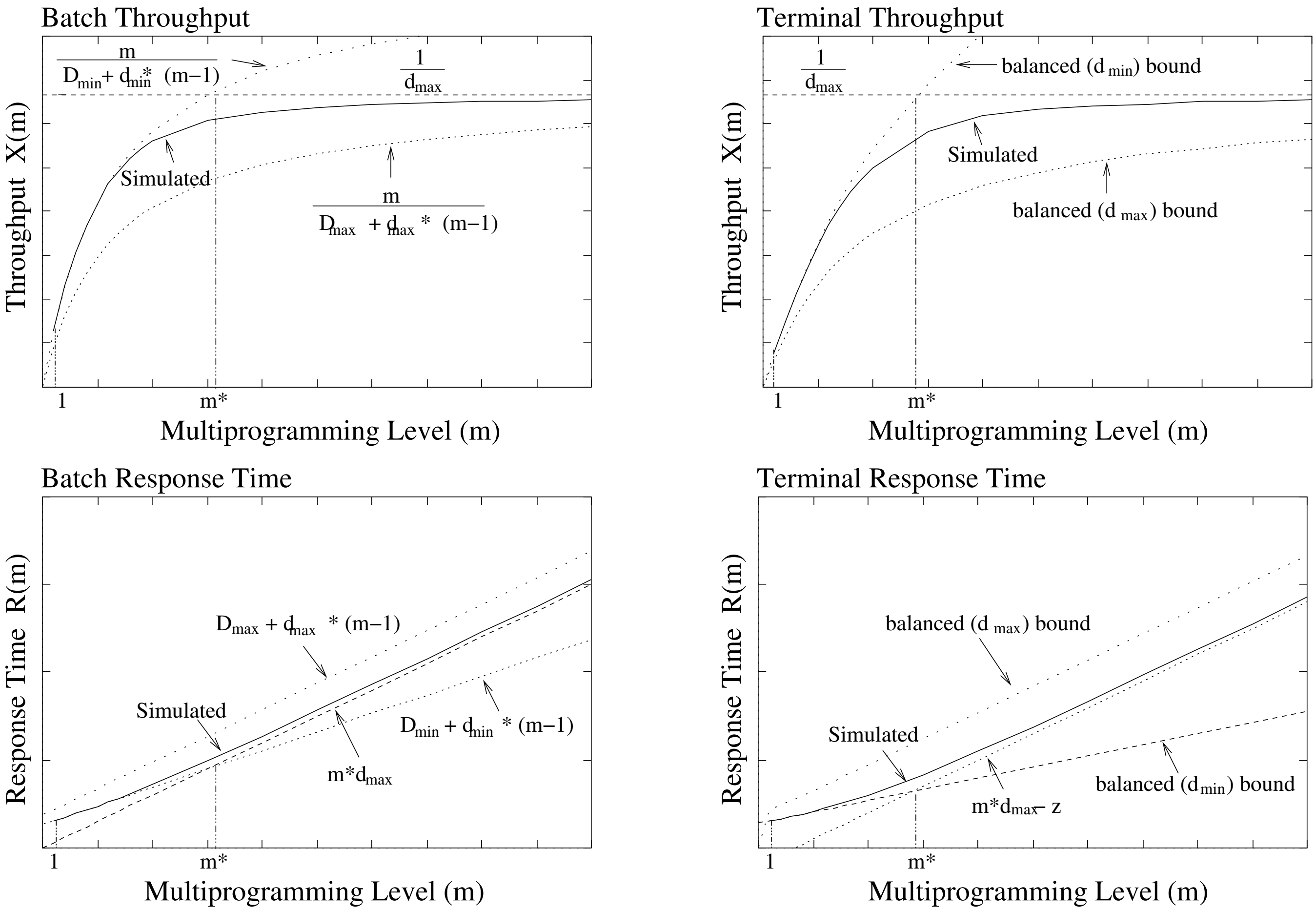

Figure 3: Balanced job bounds for exponential servers

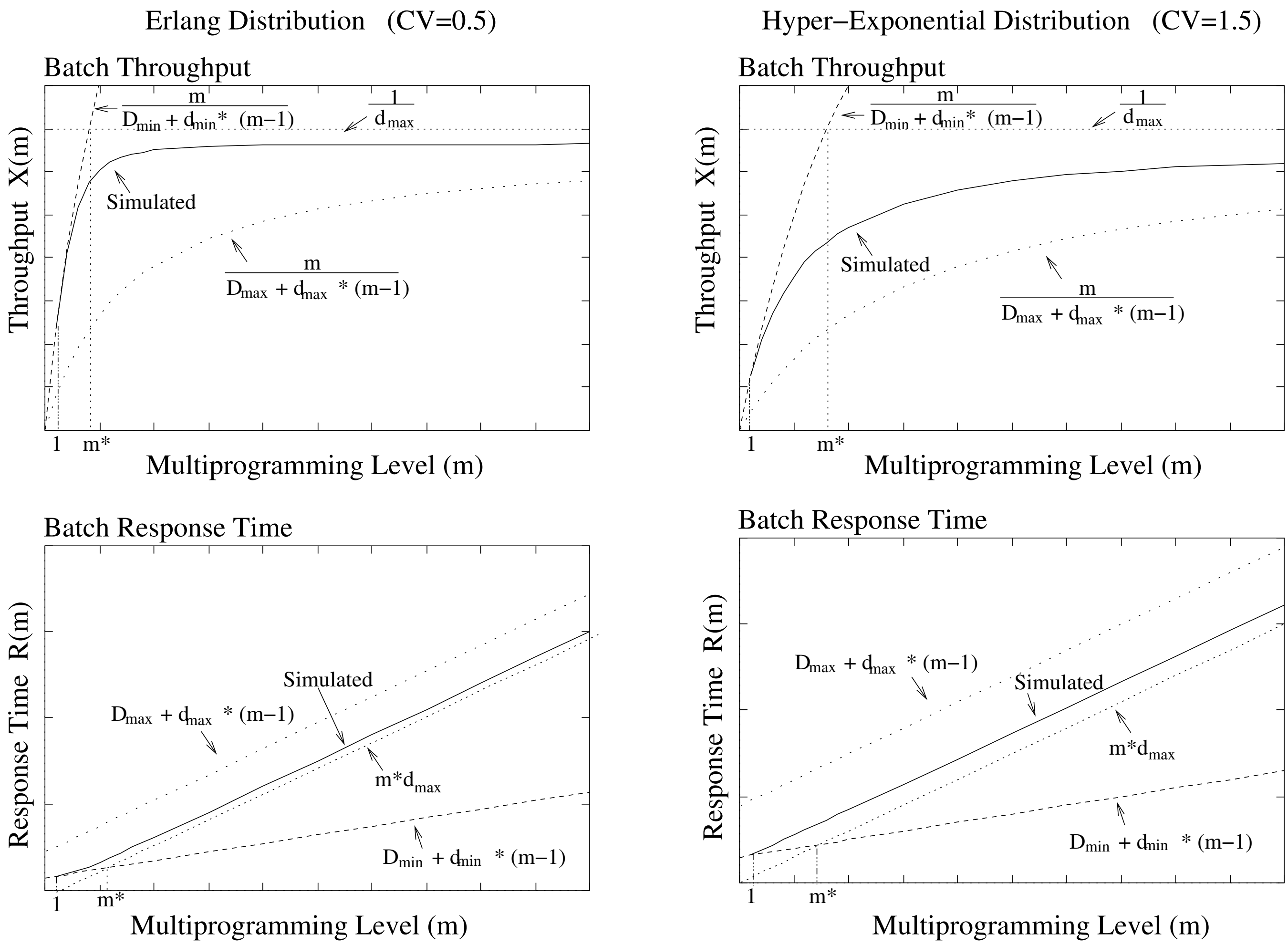

Figure 4: Balanced job bounds for non-exponential servers

and resource 4 is a parallel queue with 10 service centers and mean service time of 2 time units. In addition to the above resources, the simulated network for terminal workloads has a delay server with mean think time of 10 time units. The multiprogramming levels of the networks are varied from 1 to 50. All simulated values are accurate within 0.5 time units at 90% confidence level.

Even though Equations 2 and 3 are only proven for networks with exponential parallel queues, simulation results indicate that these equations hold for networks with Erlang and hyper-exponential distributions. Figure 4 plots the balanced job bounds for networks with all Erlang servers and for networks with all hyper-exponential servers (which have the property of the coefficient of variation being lower and greater than one, respectively). The simulated networks contain 3 resources each - resource 1 is non-parallel with mean service time of 1 time unit, resource 2 is a parallel queue with 2 service centers and mean service time of 2 time units, and resource 3 is a parallel queue with 10 service centers and mean service time of 3 time units. The simulation results show that the bounds work well for such distributions within the range of CV's simulated (0.5 and 1.5).

5 Asymptotic Bounds for Parallel Networks

The asymptotic bounding technique for parallel networks is a simple extension of the corresponding technique for non-parallel networks [24, 6]. The technique is based on the fact that performance bounds for any network can be easily computed under the extreme conditions of very light or very heavy loads. We first consider bounds for terminal workloads with M circulating jobs. Bounds for batch workloads are derived by setting the think time, Z, to zero. Under heavy load, the throughput of the system approaches $1/d_{max}$, the demand at the bottleneck center. Under light load, the bounds are computed as follows:

- In the worst-case scenario, each arriving job has to wait behind $(M-1)$ jobs at each of the resources before receiving service. The job spends Z time units thinking, $D*(M-1)$ time units waiting for service, and D time units receiving service. ($D = D_1 + D_2 + \cdots + D_K$.) The network's throughput then equals $\frac{M}{Z+M*D}$
- In the best-case scenario, each arriving job receives service immediately (*i.e.*, there are no waiting jobs). The job spends Z time units thinking and D time units in service. The network's throughput then equals $\frac{M}{Z+D}$.

The asymptotic bounds on throughput are then given by:

$$\boxed{\frac{M}{Z+M*D} \;\leq\; X(M) \;\leq\; \text{minimum}\left\{\frac{M}{Z+D},\, \frac{1}{d_{max}}\right\}}$$

The bounds on response time are derived from the throughput bounds by using Little's law:

$$\boxed{\text{maximum}\{D,\; M*d_{max}-Z\} \;\leq\; R(M) \;\leq\; M*D}$$

The optimistic light load bound intersects the heavy load bound as the multiprogramming level increases to some point, say, m^+ $(= (D+Z)/d_{max})$. After this point, the heavy load bound is tighter.

The graphs in Figure 5 plot the asymptotic bounds for a terminal and batch workload. The simulated networks have the same configuration as the networks used in Figure 3. All the bounds are straight lines (except for the pessimistic throughput/response time bounds for terminal workloads) and can be calculated manually. They can be used to quickly get a rough understanding of any system. Asymptotic bounds are useful in analyzing the effects of primary and secondary bottleneck devices [6, 7]. Several case studies showing the applications of asymptotic bounds for non-parallel networks are given in [17]. These applications also apply to asymptotic bounds for parallel networks.

6 Illustrative Example

We illustrate the application of the bounds by generating one-step performance bounds of a disk array under a synchronous I/O workload. Figure 6 shows a representation of the Hewlett-Packard SureStore E disk array FC-60 [14] that is used to validate the bounds. The FC-60 has 2 array controllers that are both connected to a single backplane bus. Each controller has 256 MB of battery backed cache memory. The backplane bus has 6 ultra-wide SCSI buses each connected to a SCSI controller. Each SCSI controller is connected to a tray. There can be up to 6 trays on the FC-60. Each tray has 2 SCSI controllers and up to 10 disks. Thus, the FC-60 holds up to 60 disks - at 73 GB per disk drive, the total raw capacity is about 4.3 terabytes. Table 1 presents the array parameters of significance to the model. All our experiments are run on one FC-60 Logical Unit (LU) containing 6 disks. The LU is configured using RAID 1/0 with a stripe unit size of 16 KB.

Table 2 presents the workload parameters of significance. The I/O workload is generated by a fixed number, M, of similar interactive jobs that each spends time at its terminal before issuing a request of size $request\_size$ to the

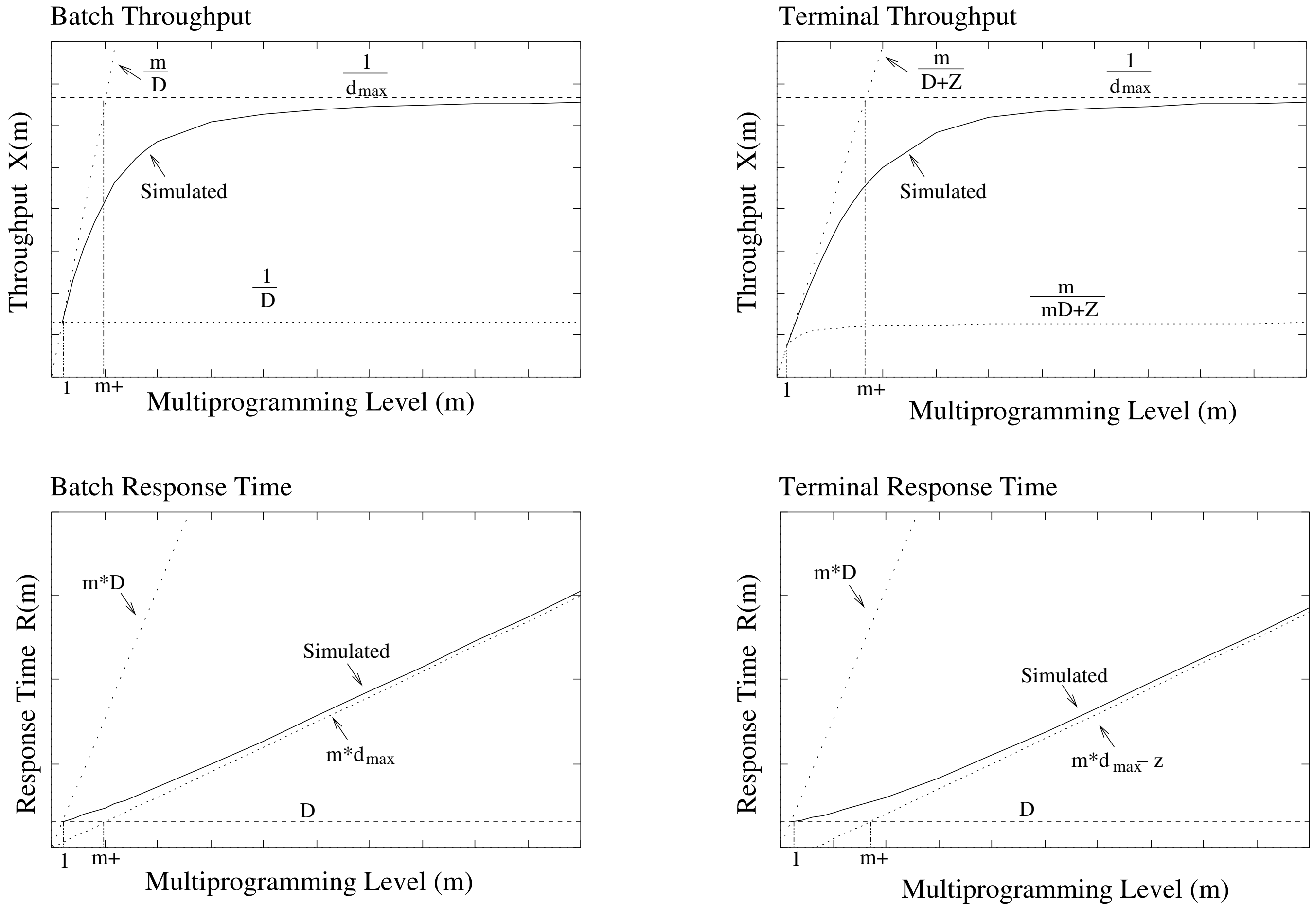

Figure 5: Asymptotic bounds for batch and terminal workloads

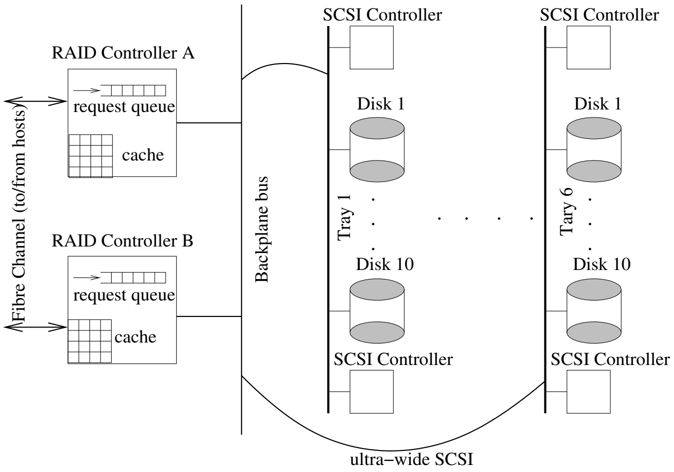

Figure 6: The HP FC-60 Disk Array

Parameter	Description	Value/Units
$cache\_size$	size of the cache at each controller	256 MB
$bus\_transfer\_rate$	mean cache transfer rate	86 MB/sec
$read\_ahead\_size$	mean number of additional bytes read from disk	$\geq$ 0 bytes
LU_disks	number of disks in a logical unit	6 disks
$stripe\_unit\_size$	size of a stripe unit	16 KB
$stripe\_size$	number of bytes in a stripe (logical row)	48 KB
$disk\_type$	type of disks in the FC-60 disk array	Cheetah73
$disk\_capacity$	total formatted capacity of a disk	73.4 GB
$disk\_avg\_read\_position\_time$	mean disk read positioning time	9.72 ms
$disk\_avg\_write\_position\_time$	mean disk write positioning time	10.2 ms
$disk\_transfer\_rate$	mean disk transfer rate	33 MB/sec

Table 1: Disk array parameters

disk array. A job is blocked while its I/O request is being serviced. Upon completion of its I/O request the job unblocks and spends some time, Z, at its terminal before submitting another read or write I/O request. The fraction of read and write requests is given by $read\_fraction$ and $write\_fraction$ $(= 1 - read\_fraction)$. The workload contains a number of consecutive requests for sequential bytes (a *run*) followed by intervals of random requests. This spatial locality of workloads is captured by the attribute run_count, the mean number of sequential requests in a run, and $random\_count$, the mean number of random requests between two runs. The temporal locality of workloads is captured by the attribute $re\_reference\_distance$, the number of bytes accessed between two accesses to the same block.

Figure 7 shows the queueing model of a disk array under synchronous I/O workloads [33]. The CPU processing time is not explicitly modeled, but is included in the terminal think time. The disk array is modeled by three components, the cache, the controller, and the disks. Even though the array cache is part of the array controller, it is modeled as a separate component since I/O requests that can be serviced by the cache need not be submitted to the disks. Components like the (SCSI) disk controllers and the interconnects are not modeled explicitly since their service times are much smaller than the disk service times and do not have a significant impact on the disk-array performance. A job, on completing its terminal think time, submits a request to the array cache. If the request can be serviced by the cache, then the cache signals service completion without forwarding the request to the controller. If the request cannot be

Parameter	Description	Value/Units
M	multiprogramming level (i.e., number of jobs issuing I/O requests)	≥ 1
Z	mean time spent by a job at its terminal	≥ 0 msec
$request\_size$	mean request size	4 KB to 256 KB
run_count	number of requests made to contiguous addresses	1-64 requests
$random\_count$	number of random requests between each run	≥ 0 requests
$read\_fraction$	fraction of read requests in the workload	0-1
$re\_reference\_distance$	amount of data accessed between consecutive accesses to the same block	bytes

Table 2: Workload parameters

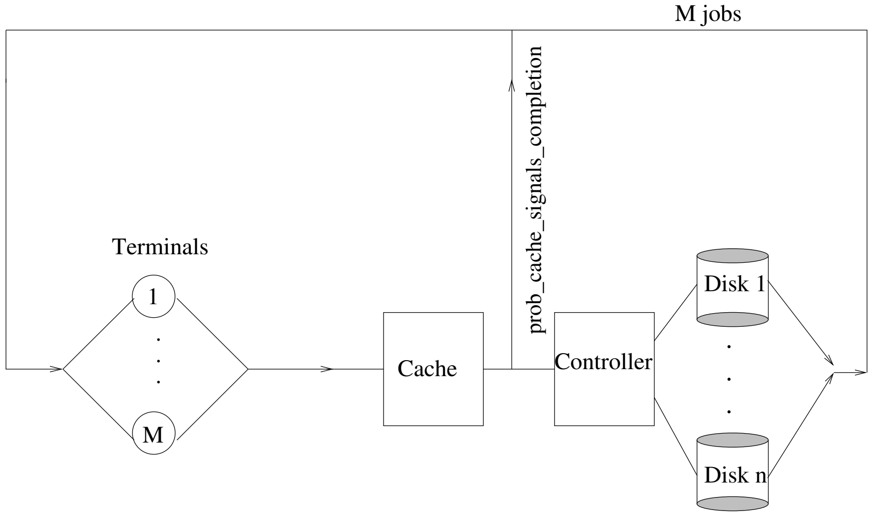

Figure 7: System Model Under a Synchronous I/O Workload

Bound Parameters	Description	Units/Value
S_1, s_1	mean cache service time per request	msec
V_1, v_1	probability that a request is submitted to the cache	1
S_2	mean controller service time per request	msec
V_2	probability that a request is submitted to the controller	$(1 - prob\_cache\_signals\_completion)$
s_2	mean disk service time per sub-request	msec
v_2	probability that a sub-request is submitted to a disk	0-1

Table 3: Inputs to bounding technique

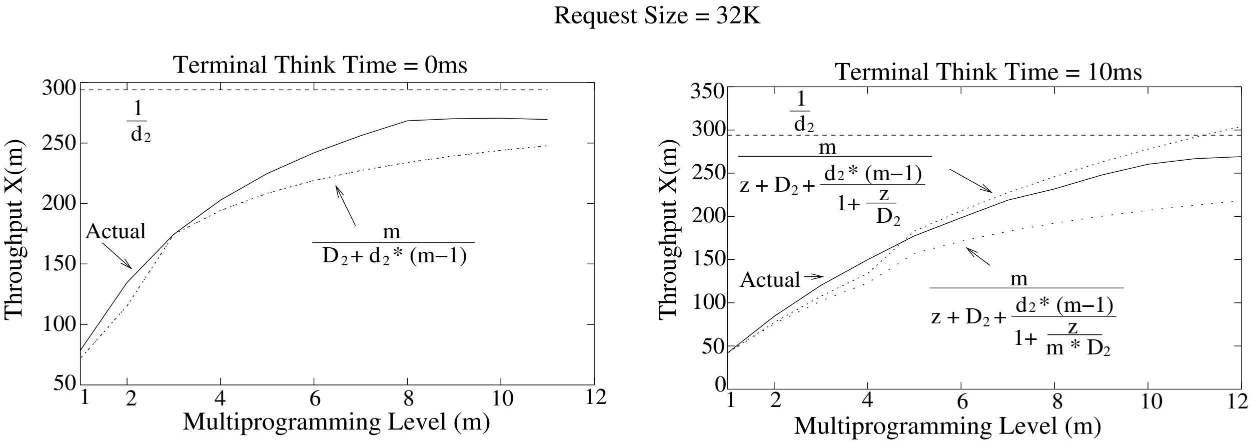

Figure 8: Random read workload

serviced by the cache, then the cache forwards the request to the controller. The controller divides the request into sub-requests and submit these sub-requests to the disks. Once all the sub-requests complete service, the controller signals service completion. Thus, the disk array is modeled by two resources - resource 1 is the cache which is modeled by a single queuing server and resource 2 is the controller which is modeled by a fork-join queue. Each disk is modeled by a queuing server of the fork-join queue. Table 3 presents the input parameters to the bounding technique.

The bounding technique is validated by comparing throughput bounds against measured throughput values from the FC-60 array for synthetic read-only workloads with request sizes ranging from 4 KB to 256 KB and varying degrees of sequentiality. The number of jobs generating requests range from 1 to 12 and the terminal think time ranges from 0 ms to 300 ms. For this synthetic workload, we compute values for the input parameters specified in Table 3 using the model presented in [33]. (The details of this disk array model are beyond the scope of this paper.) Figures 8 and 9, respectively, present the throughput balanced job bounds for two types of random read workloads and two types of sequential read workloads. For random read workloads, $prob\_cache\_signals\_completion = 0$. For sequential read workloads, $prob\_cache\_signals\_completion$ varies with the multiprogramming level. At multiprogramming level 1, the workload is highly sequential and the cache hit rate is high. At multiprogramming level $m > 1$, there are m sequential streams, so the cache hit rate decreases. Hence, the heavy load bound $1/d_{max} = 1/d_2$ is not a straight line since the visit probability to a disk varies with the multiprogramming level.

7 Conclusions

This paper presents a single-step bounding technique for parallel networks that is based on the idea that the performance of any parallel network is bounded above and below by the performance of related balanced parallel networks

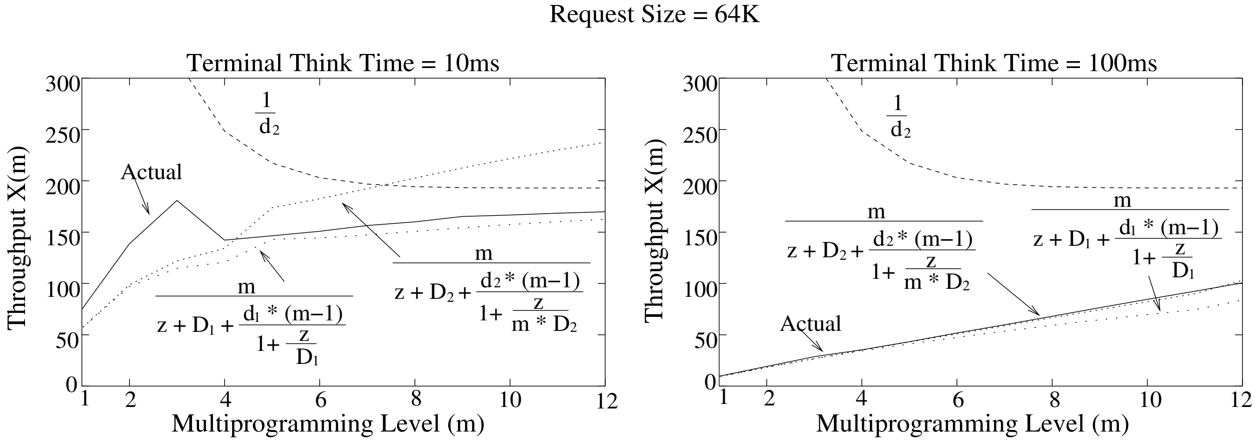

Figure 9: Sequential read workload

(balanced job bounds) and by the performance of the network under extreme workloads (asymptotic bounds). The idea behind balanced bounds and asymptotic bounds was first presented for product-form networks. The contribution of this paper is showing how this single-step bounding technique can be applied to parallel networks.

The bounding technique is derived for single-class workloads. The asymptotic bounds do not depend on the service time distributions of the modeled resources. The balanced job bounds are proven for networks where the parallel resources have service times that are drawn from an exponential distribution. Simulation results and measurements from a disk-array, however, provide limited evidence that the balanced job bounds are valid for networks with other service time distributions; more research is needed to fully investigate this proposition.

In Appendix A, we show that the proportion of the mean queue lengths at the various resources of a balanced parallel network is invariant of the multiprogramming level. This result is not used in the derivation of the bounding technique and we present it to illustrate this interesting property of balanced parallel networks. As future work, we plan to extend the bounds here to parallel networks under multiple class workloads. Another issue we plan to address is generating tighter optimistic balanced job bounds using the average demand of a resource in the network instead of the minimum demand.

A Appendix

Here, we present a property of balanced parallel networks, namely, $Q_k(M) \approx M * Q_k(1)$, for all resources k in the network. This property of balanced parallel networks is derived from the following conjecture presented in [32]:

Conjecture A.1 *The mean arrival instant queue length, A_k, at a resource k in a parallel network is approximately equal to the mean queue length, Q_k, at the resource when there is one less job in the network.*

$$A_k(M) \approx Q_k(M-1)$$

Using this conjecture and Equation 1, the response time of resources in closed parallel networks can be written as

$$R_k(M) \approx D_k + d_k * Q_k(M-1) \tag{4}$$

In [31] and [32], Equation 4 is validated against simulation results for exponential, Erlang, and hyper-exponential distributions. Equation 4 (derived from Conjecture A.1) is used to prove Result A.1.

Result A.1 *For closed balanced parallel networks,*

$$Q_k(M) \approx M * Q_k(1) \quad \forall \textit{ resources } k$$

Proof: Without loss of generality, assume that the balanced network is constructed such that $V_1 = V_2 = \cdots = V_K$ (*i.e.*, the resources in the network are visited with equal probability).
At multiprogramming level 1, $R_k(1) = D_k$ for all resources k.
By construction, $X = X_1 = X_2 = \cdots = X_K$ at all multiprogramming levels. Now, $X(1) = 1/(D_1 + D_2 + \cdots + D_K) = 1/D$, where $D = D_1 + D_2 + \cdots + D_K$.
From Little's Law, $Q_k(1) = X(1) * D_k$. It follows that,

$$Q_k(1) = \frac{D_k}{D} \qquad \text{for all resources } k \tag{5}$$

We now show that for any given multiprogramming level M, $Q_k(M) \approx M * \frac{D_k}{D} = M * Q_k(1)$ for all resources k. The result is proved by induction on the multiprogramming level.
Induction basis: The result is trivially true at multiprogramming level 1.
Induction hypothesis: Assume that the result is true at multiprogramming level $m > 1$, that is, $Q_k(m) = m * Q_k(1)$. It remains to show that $Q_k(m+1) = (m+1) * Q_k(1)$.
For any resource j in the network,

$$\begin{aligned}
\frac{Q_j(m+1)}{Q_k(m+1)} &= \frac{X(m+1) * R_j(m+1)}{X(m+1) * R_k(m+1)} \\
&\approx \frac{D_j + d_j * Q_j(m)}{D_k + d_k * Q_k(m)} && \text{from Equation 4} \\
&= \frac{D_j + d * Q_j(m)}{D_k + d * Q_k(m)} && d = d_j = d_k \quad \text{since the network is balanced} \\
&= \frac{D_j + d * m * Q_j(1)}{D_k + d * m * Q_k(1)} && \text{by induction hypothesis} \\
&= \frac{D_j + d * m * X(1) * D_j}{D_k + d * m * X(1) * D_k} && \text{from Little's Law} \\
&= \frac{D_j * (1 + d * m * X(1))}{D_k * (1 + d * m * X(1))} \\
&= \frac{D_j}{D_k}
\end{aligned}$$

Thus,

$$Q_j(m+1) \approx \frac{D_j}{D_k} * Q_k(m+1)$$

Now,

$$
\begin{aligned}
& Q_1(m+1) + Q_2(m+1) + \cdots + Q_K(m+1) = m+1 \\
\Rightarrow \quad & Q_k(m+1) * \frac{D_1}{D_k} + Q_k(m+1) * \frac{D_2}{D_k} + \cdots + Q_k(m+1) * \frac{D_K}{D_k} \approx m+1 \\
\Rightarrow \quad & Q_k(m+1) * \frac{D}{D_k} \approx m+1 \qquad \text{where } D = D_1 + \cdots + D_K \\
\Rightarrow \quad & Q_k(m+1) \approx (m+1) * \frac{D_k}{D} \\
\Rightarrow \quad & Q_k(m+1) \approx (m+1) * Q_k(1) \qquad \text{from Equation 5}
\end{aligned}
$$

Hence, $Q_k(M) \approx M * Q_k(1)$ for all resources k in a balanced parallel network.

□

Note that Result A.1 just illustrates an interesting property of balanced parallel networks. The result itself is not relevant to the derivation of the balanced job bounds. This result implies that the mean queue length (and hence, the mean response time) at each resource of a balanced parallel network can be computed quickly regardless of the multiprogramming level.

References

[1] Almeida, V.A.F., Dowdy, L.W., "A reduction technique for solving queueing network models of programs with internal concurrency", Proceedings of the 3^{rd} International Conference on Supercomputing, Boston, May 1988.

[2] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, J. Wilkes, "MINERVA: An automated resource provisioning tool for large-scale storage systems", to appear in ACM Transactions on Computer Systems.

[3] Baccelli, F. "Two parallel queues created by arrivals with two demands: The M/G/2 symmetrical case", Report INRIA, 426, July 1985.

[4] Baccelli, F., Massey, W. A., Towsley, D. "Acyclic fork-join queueing networks", Journal of the ACM, 36, 3, July 1989, pp. 615 – 642.

[5] Baccelli, F., Liu, Z. "On the execution of parallel programs on multiprocessor systems – A queueing theory approach", Journal of the ACM, 37, 2, April 1990, pp. 373 – 414.

[6] Balbo, G., Serazzi, G., "Asymptotic analysis of multiclass closed queueing networks: Common bottleneck", Performance Evaluation 26, 1996, pp. 51 – 72.

[7] Balbo, G., Serazzi, G., "Asymptotic analysis of multiclass closed queueing networks: Multiple bottleneck", Performance Evaluation 30, 1997, pp. 115 – 152.

[8] Balsamo, S., Donatiello, L., Van Dijk, N.M., "Bound performance models of heterogeneous parallel processing systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No. 10, October 1998.

[9] Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, F.G. "Open, closed, and mixed networks of queues with different classes of customers", Journal of the Association of Computing Machinery, vol. 22, no. 2, April 1975, pp. 248 – 260.

[10] Cooper, R.B. *Introduction to Queueing Theory*, third edition, Mercury Press/Fairchild Publications, MD, 1990.

[11] Flatto, L., Hahn, S. "Two parallel queues created by arrivals with two demands", SIAM J. Appl. Math., 44, Oct. 1984, pp. 1041 – 1053.

[12] Heidelberger, P., Trivedi, S. K. "Queueing network models for parallel processing with asynchronous tasks", IEEE Trans. on Computers, 31, Nov. 1982, pp. 1099 – 1109.

[13] Heidelberger, P., Trivedi, S. K. "Analytic queueing models for programs with internal concurrency", IEEE Trans. on Computers, 32, Jan. 1983, pp. 73 – 82.

[14] Hewlett-Packard Company, *HP SureStore E Disk Array FC60 User's guide*, December 2000, Pub. No. A5277-90001.

[15] Kim, C., Agrawala, A.K. "Analysis of the fork-join queue", IEEE Transactions on Computers, 38, 2, Feb. 1989, pp. 250 – 255.

[16] Kumar, A., Shorey, R. "Performance analysis and scheduling of stochastic jobs in a multicomputer system", IEEE Trans. on Parallel and Distributed Systems, 4, 10, Oct. 1993, pp. 1147 – 1164.

[17] Lazowska, E.D., Zahorjan, J., Graham, G.C., Sevcik, K.C., *Quantitative System Performance – Computer System Analysis Using Queueing Network Models*, Prentice-Hall, 1984.

[18] Lee, E.K., Katz, R.H., "An analytic performance model of disk arrays", Proceedings of ACM SIGMETRICS, 1993.

[19] Liu, Y. C., Perros, H. G. "Approximate analysis of a closed fork/join model", European Journal of Operational Research, 53, 1991, pp. 382 – 392.

[20] Liu, Y. C., Perros, H. G. "A decomposition procedure for the analysis of a closed fork/join queueing system", IEEE Transactions on Computers, 40, 3, Mar. 1991, pp. 365 – 370.

[21] Lui, J.C.S., Muntz, R.R., Towsley, D, "Bounding the response time of a minimum expected delay routing system: an algorithmic approach", IEEE Trans. on Computers, vol 44, no. 5, May 1995, pp. 1371 – 1382.

[22] Lui, J.C.S., Muntz, R.R., Towsley, D, "Computing performance bounds of fork-join parallel programs under a multiprocessing environment", IEEE Trans. on Parallel and Distributed Systems, vol. 9, no. 3, March 1998, pp. 295-311.

[23] Makowski, A., Varma, S., "Interpolation approximations for symmetric fork-join queues", Performance Evaluation 20, 1994, pp. 145 – 165.

[24] Muntz, R.R., Wong, J.W., "Asymptotic properties of closed queueing network models", Proc. 8th Princeton Conference on Information Sciences and Systems, 1974.

[25] Nelson, R., Tantawi, N., "Approximate analysis of fork/join synchronization in parallel queues", IEEE Transaction on Computers, 37, 6, June 1988, pp. 739 – 743.

[26] Thomasian, A., Tantawi, A.N., "Approximate solutions for M/G/1 fork-join synchronization", Proc. 1994 Winter Simulation conf., Orlando, Fla., Dec 1994, pp. 361 – 368.

[27] Thomasian, A., Menon, J., "Approximate analysis for fork-join synchronization in RAID5", Computer Systems: Science and Eng., 1997.

[28] Thomasian, A., Menon, J., "RAID5 performance with distributed sparing", IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 6, June 1997, pp. 640 – 657.

[29] Trivedi, K.S., *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, 1982.

[30] Varki, E., Dowdy, L.W. "Analysis of closed balanced fork-join queueing networks", Proceedings of ACM/SIGMETRICS. Also, Performance Evaluation Review, 24, 1, May 1996, pp. 232 –241.

[31] Varki, E., "Mean value analysis of fork-join parallel networks", Proceedings of ACM/SIGMETRICS. Also, Performance Evaluation Review, May 1999.

[32] Varki, E., "Response time analysis of parallel computer and storage systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 22(11), pp. 1146 – 1161, November 2001.

[33] Varki, E., Merchant, A., Qiu, X., "A performance model of disk arrays under synchronous I/O workloads", under review. (Also available at http://www.cs.unh.edu/ṽarki/)

[34] Zahorjan, J., Sevcik, K. C., Eager, D. L., Galler, B. "Balanced job bound analysis of queueing networks", Communications of the ACM, 25, 2, Feb. 1982, pp. 134 –141.